Research Quarterly

Quarterly Newsletter of the
Institute for Computer Applications
in Science and Engineering

Vol. 8, No. 4
December 1999

Editor
Shannon K. Verstynen

Editorial Assistant
Emily N. Todd

Subscription

Inside this Issue

Arcade is a web-based framework for designing and executing applications across a distributed set of heterogeneous resources. Resource management is one of the important issues in building such systems. Recently, Sun introduced the Jini connection technology for building plug-and-play networks of resources. In this article we describe our experiences with using Jini technology to build a Resource Manager module for Arcade.

Type theory is a mathematical technique widely used in computer science. In the formal methods community, type theory is at the basis of higher-order logic tools and expressive languages for formal specification. From a practical point of view, type theory has been used to improve the quality of software systems by allowing the detection of errors before they became run-time problems. This article gives a general overview of type theory and its role in the foundation of programming and specification languages.

Publication Information

Navigation Help

Past Issues

Subscription Information

You can receive e-mail notification of future editions of the **ICASE Research Quarterly** by going to the ICASE Mailing Lists Web Page and selecting the entry for the Quarterly.

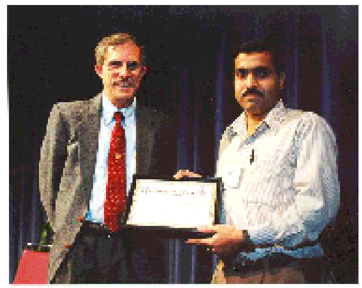
**Vol. 8, No. 4
December
1999**

Inside this Issue

Contents

ICASErs Share 1999 Bell Prize for High Performance Computer Simulation

Gordon Bell (left) and David Keyes (right)



D. Bailey (left) awarding the Bell Prize to D. Kaushik (right).

A team consisting of an ODU graduate student and faculty member - both regular research participants at ICASE - plus their collaborators at the NASA Langley Research Center and the U.S. Department of Energy shared in the Gordon Bell Prize awards on November 18, 1999 at Supercomputing’99 in Portland, OR. Their entry, entitled "Achieving High Sustained Performance in an Unstructured Mesh CFD Application," qualified for a share of the "Special" category in the annual international Bell Prize competition, which recognizes implementations of practical computational simulations that achieve new levels of performance.

The ODU/ICASE-led team of Dinesh Kaushik (a doctoral candidate in Computer Science and an ICASE graduate student), David Keyes (chairman of the Mathematics & Statistics Department, who holds an adjunct appointment in Computer Science and is an Associate Research Fellow at ICASE), Kyle Anderson (a career computational aerodynamicist at NASA Langley), and William Gropp and Barry Smith (computer scientists and developers of parallel software at Argonne National Laboratory), showed that a class of simulations previously believed to be extremely difficult on which to get good performance on a parallel computer, could, in fact, obtain within a factor of four or five of the theoretical peak performance of the world’s fastest computers. Their simulation was not the fastest overall submitted in the 1999 competition, but a three-judge panel created a special category of the award this year to recognize the significance of their achievement.

The ODU/ICASE/NASA/DOE team simulated the flow of air over an airplane wing, using 3,072 Intel Pentium Pro dual-processor nodes at a computational rate of 0.227 Teraflop/s - or 227 billion floating point operations per second. The computer they used for their highest-performing run, "ASCI Red," is located at Sandia National Laboratory in New Mexico. They ran the same cases on other high-end computers, including "ASCI Blue Pacific" at Lawrence Livermore National Laboratory in California, an SGI-Cray T3E at the vendor’s headquarters in Minnesota, an SGI-Cray Origin2000, and an IBM SP2 at Argonne National Laboratory in Illinois, ODU’s own HPC10000 from Sun Microsystems, and a variety of other machines.

In addition to building a code that implements a highly efficient algorithm portably over a wide range of high-end machines, the researchers demonstrated that such performance could be obtained from a general purpose "library" of parallel code modules written in standard high-level computer language, such as Fortran, C, or C++. In the past, writing high-performance code has often required special attention to hardware features of the computer and has been specific to one particular scientific application, so that the code is not reusable. Thanks to a multi-year investment by the Department of Energy in parallel scientific software at Argonne, the researchers were able to implement their algorithmic ideas in software that can be reused on many related applications, such as combustion, radiation transport, atmospheric and ocean modeling, petroleum reservoir modeling, or semiconductor device simulation.

In presenting the work to the audience at Supercomputing’99, Keyes claimed that many scientific codes written for earlier generations of computers can be adapted, with a reasonably small investment of human expertise, to run on parallel computers following the paradigm employed on their transformed NASA code. The memory of such computers is usually distributed over hundreds or thousands of processors, and a processor running one copy of a legacy program cannot transparently access the memory of another, which may contain data that it requires. Using the library developed by the Argonne collaborators, this important task becomes transparent.

To be effective for supercomputing an algorithm must have three properties: it must run fast on a single processor, it must "scale" efficiently to many processors (for instance, it should run nearly 1000 times faster on 1000 processors than it does on one), and it must converge nearly as quickly on a large number of processors as on one. (Running fast and converging quickly are different concepts. Parallelization may change an algorithm so that even though it runs fast in the sense of many operations per second, it also requires many more operations to complete, defeating the former gain.) If any one of these three criteria are left out, it is known how to excel at the other two. The ODU/ICASE/NASA/DOE team worked methodically and made practical contributions on each of the three criteria in an effort that has been ongoing in a general sense with ICASE support for several years.

A particular difficulty faced by the researchers was that the NASA code, originally written by Anderson, makes use of unstructured grids. Whereas fields stored on structured grids have neighbors in regular positions, which can be directly addressed in standard computer languages. An address table must be consulted on unstructured grids in order to find the neighbors of a given field unknown, an operation that needs to be performed thousands of times for each unknown in the course of solution. The resulting address translations can "choke" the flow of data between memories and processors. Nevertheless, the flexibility of unstructured grids is desired in many contemporary applications. The researchers could not do away with the address translation stage, but they clustered and ordered the data defined on the grid in such a way that a single translation step could take the place of many translations in the previous code.

The specific collaboration that led to this year’s award was begun during an interagency workshop on unstructured grid techniques co-organized by Smith and Keyes at Argonne in September 1996, which was supported in part by NASA Langley’s HPCC program and attended by all five of the collaborators. Kaushik immediately thereafter made the project the subject of his doctoral dissertation. He expects to complete work during the upcoming calendar year and assume a post-doctoral position at Argonne, where he has been physically resident during all of 1999.

The "ASCI" machines, on which all but one of this year's Bell Prize winning entries were run, derive their names from the Department of Energy's Accelerated Strategic Computing Initiative, a research program whose goal is to show that the testing of nuclear weapons may be replaced by computer simulations. The scientific support provided by this program is an important component of the Comprehensive Nuclear Test Ban Treaty, which the Senate rejected in its present form earlier this fall. It has not yet been convincingly demonstrated to many scientists and policy makers that computational simulations alone can provide enough information to maintain and monitor the nuclear stockpile in the absence of tests. The reliability of such simulations is related to the speed at which they can be carried out. As demonstrated at Supercomputing'99, strong steps in this direction were made during the past year. In contrast, there were no directly ASCI-affiliated Bell Prize winners in 1998, even though three ASCI computers, costing taxpayers upwards of $100 million, were already then operating in production mode at DOE laboratories.

Old Dominion University is one of 18 universities nationwide participating with the DOE at the level of a one-million dollar contract or more in the ASCI research endeavor. Besides Keyes, Professors Alex Pothen and Linda Stals of Computer Science are intensely involved in the program. Dr. Dimitri Mavriplis, an ICASE Research Fellow, is also an integral member of the project. Mavriplis's aerodynamic simulation codes have run with high efficiency on "ASCI Red" and "ASCI Blue Pacific" using grids even larger than the grids of this year's prize-winners.

The Gordon Bell Prize was instituted in 1988 by Dr. Gordon H. Bell, one of the designers of the DEC Vax computer systems, and a long-time personal patron of the field of high performance computing. Technical Paper entries to the Bell Prize competition are judged by a panel of respected figures in high performance computing. The principal prize is for total sustained performance, with additional prizes in price-performance, speed-up, and compiler speed-up being awarded from year to year as appropriate. The four prize-winning teams of 1999 split a total of $5000 in prize money.

The team with which the ODU/ICASE-led team shared this year's "Special" category is a separate Argonne National Laboratory effort, headed by former ICASE consultant Paul F. Fischer. It is also related to fluid flow. The team that won the total sustained performance award is composed of 13 members from Lawrence Livermore National Laboratory, the University of Minnesota, and IBM. The team that won the price-performance award is from Japan.

Old Dominion University and ICASE were honored in another way in the Supercomputing'99 awards process. Graduate student David Hysom of Computer Science, a doctoral candidate being supervised by Alex Pothen, was one of four finalists (internationally) in two prize categories, together with Pothen: Best Paper and Best Student Paper. Hysom is a graduate student intern at ICASE and Pothen has been a regular consultant to ICASE since 1994. Ultimately, these Best Paper awards were won by others, but Hysom's talk on the "Efficient Parallel Computation of ILU Preconditioners" was well-attended and led to many audience interactions. The Hysom-Pothen project is closely related to the Kaushik-Keyes-NASA-DOE project. The algorithms Hysom is developing as part of his dissertation are likely to be incorporated into such simulations in the future, replacing one of the weakest components of the current procedure and further improving the efficiency of parallel computing for this class of simulations.

Though they were not co-authors of the winning particular submission, two other developers of the DOE software library at the heart of the winning also have ODU/ICASE connections. (The software, known as the Portable Extensible Toolkit for Scientific Computing, or "PETSc," was the subject of an ICASE short course in December 1996 and a tutorial co-sponsored by ICASE at Parallel CFD'99 last May.) Lois Curfman McInnes, who did a three-semester post-doc as an ODU employee under Keyes and Pothen on an NSF "Grand Challenge" grant, is a third member of the Argonne team that produced the PETSc. McInnes earned her Ph.D. from the University of Virginia. Satish Balay, presently a staff computer scientist at Argonne and the fourth member of the team, graduated with an M.S. in CS from ODU in 1995, as one of Keyes's first ODU research students. The PETSc project web page is http://www.mcs.anl.gov/petsc.

Keyes and Anderson met when Keyes used to consult at NASA Langley through the Institute for Computer Applications in Science and Engineering during summers of the period 1986-1993, before Keyes joined ODU. Anderson's code is named "FUN3D" for "Fully Unstructured -- Three Dimensional" (pronounced Fun-3-D). Its web page is http://fmad-www.larc.nasa.gov/~wanderso/Fun

ICASE Research Quarterly
Quarterly Newsletter of the Institute for
Computer Applications in Science and Engineering

Vol. 8, No. 4
December 1999

Inside this Issue

ICASE Welcomes New Staff and Visiting Scientists

ICASE has recently welcomed several new staff members and visitors, including Staff Scientist Luc Huyse and Visiting Scientist Yeon-Gon Mo, whose profiles are provided below:

Dr. Luc Huyse joined ICASE as a Staff Scientist in Applied and Numerical Mathematics in October 1999 and is working in the area of robust optimization under uncertainties.

Luc graduated from the Katholieke Universiteit Leuven in Belgium as a Civil Engineer in 1991 with an honors thesis in the field of traffic flow modeling. From 1991 till 1994 he worked in two areas: finite element modeling of fiber-reinforced concrete and the assessment of structural damage in concrete structures as well as in historic masonry buildings.

Because an appropriate statistical model of the uncertainties is paramount in all these applications, he started graduate school at the University of Calgary in Canada to study risk and reliability analysis where he obtained a M.Sc. in 1995 and a Ph.D. in 1999, both in Civil Engineering.

His work is concentrated in the areas of robust estimation of extreme values, reliability-based design, random field modeling and the Stochastic Finite Element Method.

Dr. Yeon-Gon Mo joined ICASE as a Visiting Scientist in October 1999.

Yeon-Gon received his Ph.D. in Electrical Engineering at the University of Nebraska in 1999. While at the University of Nebraska, he worked as a Research

Assistant. His dissertation project concentrated on the material processing, synthesis, and characterization of photochromic and thermochromic materials including ceramics, polymers, and organic-inorganic composites using sol-gel processing, CVD, and magnetron reactive sputtering. He was involved in the NASA project on "Thermal Control Thin Film Coatings."

His current research work focuses on networked rectenna array for smart material actuators and thin-film microcircuit embodiment of PAD and VUC circuits.

ICASE Research Quarterly
Quarterly Newsletter of the Institute for
Computer Applications in Science and Engineering

**Vol. 8, No. 4
December
1999**

Inside this Issue

Using Jini™ Technology for Distributed Resource Manager in Arcade*

A. Al-Theneyan
Old Dominion University and ICASE
P. Mehrotra
ICASE
M. Zubair
Old Dominion University and ICASE

Arcade

Arcade is a web-based integrated framework being built to provide support for a team of discipline experts to collaboratively design, execute, and monitor multidisciplinary applications on a distributed heterogeneous network of workstations and parallel machines [1].This framework is suitable for applications, which in general consist of multiple heterogeneous modules interacting with each other to solve the overall design problem, such as the multidisciplinary design optimization of an aircraft.

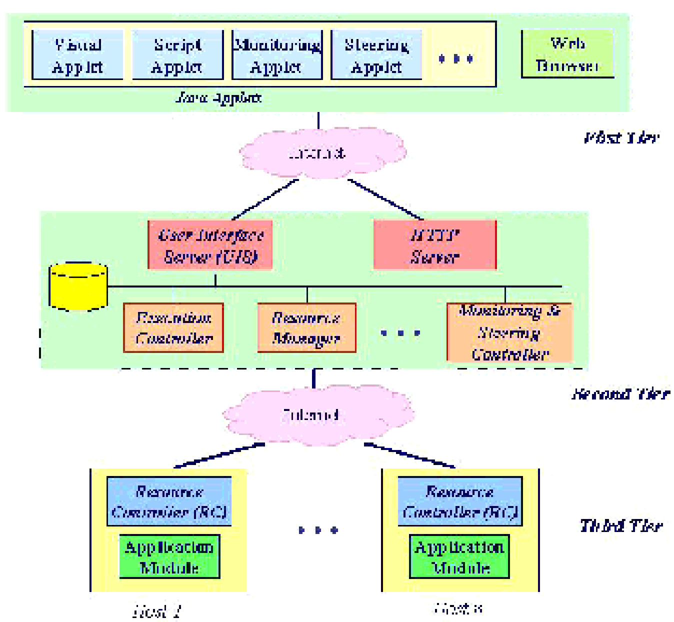


**Figure 1: Architecture of the Arcade System
(Click on figure to view enlarged image.)**

Navigation Help

Past Issues

As shown in Figure 1, Arcade is based on a three-tier architecture. The front-end is a web-based, lightweight client which provides the user interface to the whole system. It consists of applets which allow users to design an application, monitor and allocate resources, and execute, monitor and steer the application in a collaborative manner. The back-end consists of the distributed resources that are used to actually execute the user modules and application codes. A lightweight controller executes on each resource providing a gateway to the resource.

Most of the logic of the system is contained in the Java-based middle tier. Among other modules, the middle-tier consists of the user interface server which provides logic to process the user input, the execution controller which manages the overall execution of the application, the data manager which controls the data, and the application monitoring and steering controller which services the monitoring and steering request from the clients.

One of the major components of the middle tier is the *Resource Manager (RM)*. The RM keeps track of the distributed resources which comprise the execution environment and provide information about these resources to the client/user upon request. In a distributed environment, such as the one envisioned for Arcade, the RM has to be flexible, extensible, and dynamic. The resources are varied in nature ranging from compute engines, to data servers, to specialized instruments. The RM has to be flexible enough to not only handle such heterogeneity but also provide information about these systems. The environment is generally dynamic in which systems randomly join and leave. The RM should be able to handle such dynamic behavior without human intervention. Also each resource generally has some static characteristics, e.g., the speed of the CPU on a compute engine, along with some dynamic attributes, e.g., the current load on a machine. The RM should keep track of not only the static information but also the dynamic information. Another issue is that Arcade is targeted to work in a multi-domain environment, accessing and utilizing physically distributed resources spread across separately controlled and managed domains. We have been investigating the use of the recently introduced Jini connection technology to build the Arcade Resource

Manager and in this article we describe our experiences and the enhancements we have had to make to use this technology in Arcade.

Jini in a Nutshell

Jini connection technology [2] from Sun Microsystems, provides simple mechanisms for resources to join together in a federation with no human intervention. The resources, both hardware and software, can provide services to clients on the network. The Jini connection technology, based on Java, provides the necessary protocols for services to register themselves with lookup services and for clients to then discover these services.

The whole technology can be segmented into three categories: infrastructure, programming model and services. The infrastructure includes lookup services that serve as a repository of other services and an extended version of Java based RMI (Remote Method Invocation), which defines the mechanism of communication between the members. The programming model includes interfaces such as discovery, lookup, leasing, remote events and transactions which ease the task of building distributed systems.

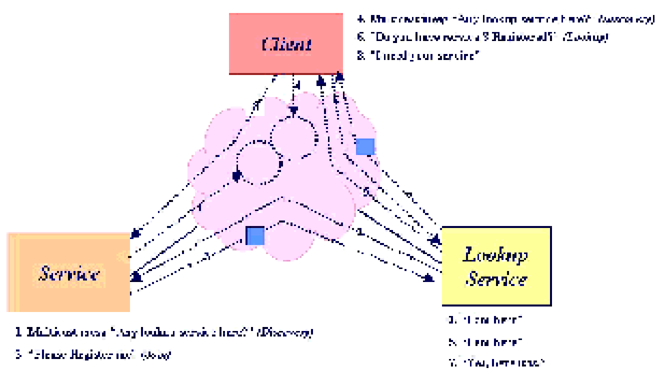


Figure 2: Sequence of steps required to use Jini Technology
(Click on figure to view enlarged image.)

A service is a central concept within Jini. It is essentially an entity that can be used by a person, program or another service to perform a required task. The runtime infrastructure supports the discovery and join protocol that enables services to discover and register with lookup services. *Discovery* is the process by which anentity locates lookup services on the network and obtains references to them. *Join* is the process by which a resource registers the services itoffers with lookup services. In particular, the resource may post a service object with the lookup service. Such a service object, contains the Java programming language interface for the service including the methods that users and applications will invoke to execute the service, along with any other descriptive attributes.

On the other hand, clients use the same protocol to locate and contact services. Thus, the discovery protocol isused to locate lookup services. Once an appropriate lookup service has been found, the client can query it to find the reference to the service that it requires. A service is located by matching its type - that is, by its interface written in the Java programming language - along with descriptive attributes provided in the object. A wild card mechanism allows some flexibility in this matching process. A client may then download the posted service object in order to directly use the service. At this point, the Jini Lookup service is no longer part of the picture, the client and the service interact directly based on the methods provided within the service object. Figure 2 shows a simplified version of the sequence of steps that take place for a service to discover and join a lookup service and for a client to use the lookup service to locate and interact with the service that it is seeking.

Jini supports a leasing mechanism that allows the federation to be flexible and resilient to failures. A service only leases an entry in the lookup service for a negotiated amount of time. When the time expires, the service has to re-register itself with the lookup service. Such a protocol allows the network of services to degrade gracefully. For example, if a service goes down without notification, its entry will expire after the leased time leaving no trace of it in the federation. When the resource returns it can re-register itself making its service available for client use.

The Jini distributed event mechanism allows client and services to be notified when pre-specified events occur. Thus, for example a client may request that a lookup service notify it when a certain type of service comes online. This allows clients and services to wait for events without having to periodically pool the lookup service.

Using Jini in Arcade

We have a built a simple resource monitor and manager in Arcade using the Jini connection technology. We have defined our own Java-objects to represent the resources, workstations, and parallel machines in particular. When the Arcade environment comes online, it starts an Arcade Resource lookup service on a designated workstation. As the resources in the environment come online a Resource Controller is started up on the resource. This controller then discovers and joins the Arcade Resource lookup service and uploads its service object. The service object contains some static information, e.g., the type of machine, the speed of the cpu, memory size, etc. It also contains dynamic information, such as the current load. At this point the object is designed such that the dynamic information is put into the service object by the resource itself. We are looking at the possibility of actually loading the object with code which could be invoked by the client to generate the information on the fly.

Our experience with Jini technology has brought out some problems in using the technology for resource monitoring such as the issues of scalability, security, no support for range queries, etc. Here we concentrate on one specific problem: using Jini across networks that do not support multicasting. Jini uses multicasting across a network for two if its internal protocols. The Multicast Request Protocol is used by discovering entities (services/clients) to locate all the nearby lookup services currently active in the environment. Once discovered, the discovering entity and the lookup service communicate with each other using unicast protocols since they each know the other's IP address and port number. The Multicast Announcement Protocol is used by lookup services to periodically announce their presence on the network. This is useful in situations where a new lookup service comes online or when a lookup service is restored after a failure.

Such multicast messages work fine in environments which support multicast. However, some routers on the Internet do not support routing of multicast packets for a variety of reasons. Also, some organizations are not willing to open their firewalls to multicast so as to avoid security problems. Similarly, a local area network divided into subnets, may disable multicast traffic across the subnets to avoid unnecessary traffic that may result in performance degradation. This blocking of multicast traffic across subnets prohibits the use of Jini in such an environment.

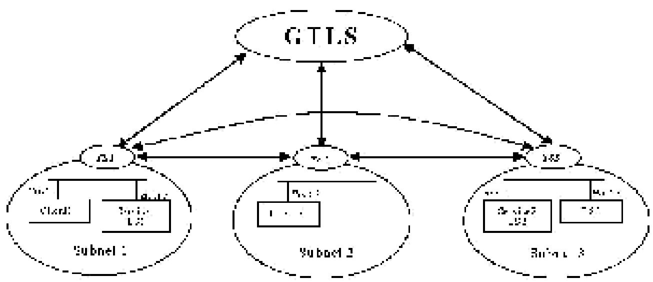


Figure 3: Tunnel Service required for non-multicastable networks
(Click on figure to view enlarged image.)

To solve this problem, we have used the well known concept of tunneling of messages across the subnets. As shown in Figure 3, a lightweight *Tunneling Service (TS)* is introduced on each subnet. Each TS listens for multicast announcements and requests in its subnet. When it receives such a message, it wraps it up and *tunnels* to all the other TSs in the system. On the other hand, when it receives a tunneled message from one of the other TSs, it unwraps the message and multicasts it within its own subnet. This allows messages multicast in one subnet to be sent across networks. Note that as result of such a broadcast two entities in different subnets may wish to communicate with each other. Since such communication is generally based on direct unicast messages, the tunneling mechanism is not involved in it.

In order for the system to work properly, each of the TSs needs to know about all the other TSs in the environment. Thus, we need a central repository that keeps track of all the currently active TSs. Jini provides the functionality required for just such a repository. Hence, we implemented this repository as a Jini lookup service called the *Global Tunneling Lookup Service (GTLS)*. As a TS is started, it registers itself with GTLS uploading a proxy object which can be used to communicate with it directly. Using the distributed events interface of Jini, every TS can get notified when a new TS joins or leaves the system. Each TS maintains an internal list of currently active TSs in the environment and uses it to broadcast tunneled messages. In our implementation, since each TS relies on the unicast discovery protocol in all its interactions with the GTLS, it needs to know the IP address and the port of the GTLS.

Note that the GTLS can also be implemented as a stand alone service without using Jini technology. In such a case, the TSs do not need to know about each other and just forward the message to be tunneled to the GTLS. The GTLS maintains the list of active TSs and whenever it receives a message broadcasts it to the other TSs. Each TS still has to listen for incoming tunneled messages which it multicasts in its subnet. This central directory approach has some advantages over the Jini-based approach in that the TSs are lighter weight. However, in this approach, we cannot take advantage of Jini's event notification and leasing mechanism (unless we build it ourselves) and also the GTLS may become a bottleneck.

Jini Modifications

We had hoped to implement our Jini-based system without making any modifications to Jini, i.e., without making any changes to the behavior of the clients, services or lookup services. However, in order to overcome some of the obstacles of tunneling, we have had to modify the format of the outgoing request and announcement messages. Note that only the message formats need to be modified, the behavior of the rest of Jini remains intact and does not need to be changed.

The first problem is the issue of loopback of a message which a TS receives from outside and then multicasts in its own subnet. If we do not make any modification, the TS will receive this message as a multicast message in its own subnet and try to tunnel it out to the other TSs. Thus, we had to introduce a flag in the message header which allows a TS to distinguish a multicast message originating from the local subnet from a message received from outside.

The second problem is that of the host address of the sender in a tunneled request message. When responding to a request message from a discovering entity, a lookup service uses the port number included in the message. However, it obtains the IP address of the sender by inquiring about the source of the multicast message. This works well within a subnet where the multicast message is originating from the discovering entity itself. However, in the case of a tunneled request, the IP address is going to represent the TS's host and not the host of the original discovering entity. To overcome this problem, we have added the IP address of the host of the sending entity in the header of the request message. We do not need to add it in the announcement message since it already contains the host's IP address.

We have implemented the system, as described above, using JDK 1.2 and Jini reference implementation 1.0. We have used it to successfully tunnel messages between subnets in a single domain and also between the icase.edu and the cs.odu.edu domains.

Future Work

We are currently in the process of enhancing the capability of the resource monitoring and management subsystem. In particular, we are investigating the use of an XML-based resource specification in order to gain the flexibility provided by XML. More information on the Arcade system can be found at http://www.icase.edu/arcade.

References

[1] Z. CHEN, K. MALY, P. MEHROTRA, and M. ZUBAIR. ARCADE: A Web-Java Based Framework for Distributed Computing, WebNet 99, October 1999.

[2] Jini Connection Technology, http://www.sun.com/jini.

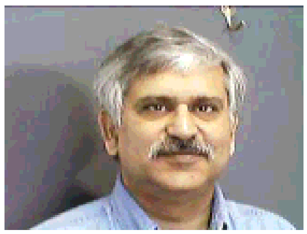
ICASE Research Quarterly
Quarterly Newsletter of the Institute for Computer Applications in Science and Engineering

Profile

**Vol. 8, No. 4
December
1999**

Inside this Issue

Piyush Mehrotra



Dr. Piyush Mehrotra is a Research Fellow at the Institute for Computer Applications in Science and Engineering, located at NASA Langley Research Center in Hampton, VA. His research interests include languages, compilers and runtime support systems for massively parallel and distributed heterogeneous environments. In particular, he has been involved with the development of several parallel languages including Kali and Vienna Fortran and has actively participated in the design of High Performance Fortran, a standard language for data parallel programming. He is involved in developing a Web-based framework for supporting distributed computing over a heterogeneous network of workstations and parallel machines. He has several publications in these areas.

Dr. Mehrotra is currently leading Computer Science Research including the HPCCP System Software effort at ICASE, which involves research in all aspects of support software for parallel computing. Before coming to ICASE, he taught at the Department of Computer Science at Purdue University for four and one-half years. He received his Ph.D. in Computer Science from the University of Virginia in 1982.

ICASE Research Quarterly
Quarterly Newsletter of the Institute for
Computer Applications in Science and Engineering

Vol. 8, No. 4
December 1999

Inside this Issue

Type Theory and Its Applications in Computer Science

César Muñoz
ICASE

Introduction

In elementary school we were taught that a *set* is a collection of elements having some characteristics in common. Later, teachers asked us whether or not an element belongs to a given set. So, for example, they stated that the *collection of stars in the universe* is a set, and then they asked us if the element *moon* belongs to that set or not. The very simple theory of sets behind these two concepts:

1. every predicate is a set, and
2. it is always possible to ask if an element satisfies a predicate,

is known in our days as the *naîve set theory*. In 1902, Russell has shown that the naîve set theory is inconsistent, i.e., it leads to paradoxes [23]. A well-known paradox due to Russell is the following. Let *A* be the set of all the sets that are not elements of themselves. Is *A* an element of itself? Both positive and negative answers to this question raise a contradiction.

To solve Russell's paradox, two theories were proposed: the Set Theory of Zermelo-Fraenkel [6] (a.k.a. set theory) and the Type Theory of Whitehead-Russell [23] (later simplified by Ramsey and Church [19, 4]) (a.k.a. type theory).

In set theory the postulate that every predicate is a set has been replaced by very precise rules of construction of sets. For instance, the comprehension rule to define a set by a predicate is only allowed over previously constructed sets. In set theory, the "set" of all the sets that are not elements of themselves is simply not a set since it does not respect the comprehension rule.

On the other hand, in type theory the postulate that allows us to ask if an element belong to a set has been constrained. In this approach, mathematical objects are stratified in several categories, namely *types*. For instance, a set has a different type from its elements. Since all the elements of a set must have the same type, the question "Is *A* an element of itself?" is not a valid question. From the type theory point of view, teachers are not always allowed to ask about arbitrary elements on arbitrary sets.

Set theory (together with classical logic) is the standard foundation of modern mathematics. However, type theory, and all its variants, is widely popular in the computer science community. In the rest of this article we give a general overview of type theory and its role in computer science, i.e., the foundation of programming and specification languages.

Type Theory and Its Applications

A minimal type system, known as the *Simple Type Theory*, was proposed by Church in [4]. In that theory, mathematical objects are of two kinds: *terms* and *types*. The terms of the Simple Type Theory are the terms of the lambda-calculus, itself being a formalization of partial recursive functions proposed by Church [5]. Types can be basic types or functional types *A* --> *B* where *A* and *B* are types.

In lambda-calculus, terms can be variables, functions, or applications. Only terms that follow a type discipline are considered to be valid. The type discipline is enforced by a set of *typing rules*. A typing rule says, for example, that a function *f* can be applied to a term *x* if and only if *f* has the type *A* --> *B* and *x* has the type *A*. In that case, the application *f(x)* has the type *B*. Thanks to the typing rules, Russell’s paradox cannot be expressed in the simple type theory.

Type checking is decidable in the simply typed lambda-calculus. That is, it is decidable whether or not a term has a type, in a given context, according to the typing rules.

The Simple Type Theory can be extended straightforward with types for Cartesian products, records, and disjoint unions [3]. For this reason, simple types have been largely exploited by designers of programming languages. Indeed, most of the modern programming languages support, to some extent, a simply-typed system. In these languages, type checking, implemented by the compiler, is a powerful tool to reject undesirable programs. In other words, programs violating the type discipline are considered harmful, and therefore, they should be rejected by the compiler. For instance, in Pascal [10], boolean functions cannot be applied to integers. This restriction happens to be a simply typed rule enforced by the compiler. On the other hand, C [11] supports a more liberal typing discipline; the compiler warns of some violations, but it seldom rejects a program. Almost every C-programmer knows the danger of this flexibility in the language.

When writing formal specifications, the choice of a *typed* language, in opposition to a language based on set theory, is not always evident [12]. In contrast to programs, specifications are not supposed to be executable. Thus, a too restrictive type theory, as for example the simply typed lambda-calculus, quickly becomes cumbersome to write enough abstract specifications. Several variants of type theories have been proposed in the literature, most of them are still convenient to write formal specifications and powerful enough to reject specifications that are undesirable [21]. Let us review some extensions to the Simple Type Theory.

Polymorphism and Data Types

A major improvement to the Simple Type Theory, is the System F proposed by Girard in [7]. System F extends lambda-calculus with quantification over types; that is, it introduces the notion of *type parameters* which is at the basis of the concept of *polymorphism*. In this system, generic data types as list, trees, etc. can be encoded. Type checking is still decidable in a type system that supports polymorphism and abstract data type declarations.

In programming languages, polymorphism allows the reuse of code which works over structures parameterized by a type. For instance, a sort function is essentially the same whether it works over a list of integers or a list of strings. Polymorphic-typed languages exploit this uniformity without losing the ability to catch errors by enforcing a type discipline.

Although most of the specification languages based on higher-order logic support polymorphism [17, 13, 18, 2], just a few modern programming languages implement it correctly.

The functional programming languages of the ML family [16] are strongly typed languages that support algebraic data types and polymorphism. They use the Milner's type inference algorithm [15]. That is, although the language is strongly typed, the types of all the expressions occurring in the program are inferred by the compiler. Hence, ML programs are almost free of type declarations.

Object-oriented imperative languages as Eiffel [14], C++ [22], and Java [1] support polymorphic types to some extent. However, object-oriented features, side effects, and polymorphism result in very complex type systems. Eiffel uses a rather complicated and ad-hoc type system, C++ follows the same liberal discipline of C with respect to type checking, and polymorphism in Java is restricted to single inheritance.

Dependent Types and Constructive Type Theories. Dependent types is the ability to define types which depend on terms. For instance, in Pascal the type declaration `array[1..10] of integers` is a dependent-type declaration since it depends on expressions `1` and `10`. A general theory of dependent types, called LP, was proposed by Harper et al. in [9].

Dependent types have been extended with polymorphism and inductive data types in a very expressive extension to the lambda-calculus called the *Calculus of Inductive Constructions* (CIC). This calculus is the logical framework of the proof assistant system Coq [2]. Type checking is decidable in CIC and the calculus satisfies the strong normalization property, i.e., functions defined using the CIC formalism always terminate. The Calculus of Inductive Constructions supports the *propositions-as-types* principle of the higher-order intuitionistic logic. According to this principle, a proof of a logical proposition A is the same as a term of type A. This isomorphism between proofs and terms is also known as the *Curry-Howard* isomorphism [8]. In practice, the Curry-Howard isomorphism is used to extract a program from the constructive proof of the correctness of an algorithm. Programs extracted in this way satisfy the termination property.

Although very simple dependent types are used in most programming languages, general dependent types and constructive types are still hard to handle in programming languages.

Subtyping and Other Mysteries. In type theory every object has at most one type. A drawback of this postulate is that an object as the natural number *1* has to be different from the real number *1*. A way to handle this problem is to introduce *predicate subtyping* [21], i.e., the ability to define new types by a predicate on previously defined types. For instance, the type *nat* can be defined as a subtype of *real* such that it contains only the numbers generated from *0* and *+1*. Via predicate subtyping the natural number 1 is also a real number.

The type theory of the general verification system PVS [20] supports predicate subtyping. Unfortunately, general predicate subtyping rends type checking undecidable. In PVS, the type-checker returns a set of type correctness conditions (TCCs) that should be discharged by the user; these TCCs guarantee the type correction of the formal development. In practice, TCCs are not a problem since PVS provides a powerful theorem prover which implements several kinds of decision procedures and automation tools.

Due to the undecidability problem, general predicate subtyping is not used in programming languages. However, object-oriented programming languages opened the door to another interesting kind of subtyping: inheritance. Via inheritance, two structurally different types may share some elements. Related concepts to inheritance are those of *overloading*, that is, the ability to use the same name for different functions, and *dynamic typing*, that is, the ability for objects to change their types during the execution of a program. The formal semantics of all these concepts in a typed framework is still subject of research and controversy.

Summary

Type theory offers a convenient formalism to write specifications and the ability to reject undesirable specifications long before they were refined into actual implementations. In programming languages, type checking allows us to catch potential run-time errors at the compilation phase. Type systems used in specification languages and in programming languages differ in complexity and in expressiveness. Current research in the area includes bringing the benefits of very expressive type systems to programming languages used by practitioners. In order to do that, it is necessary to adapt and simplify the highly mathematical notations of the complex type systems into easily handled programming language features.

The Formal Methods Group for Aviation Safety at ICASE conducts basic research on the application of type theory techniques to improve the safety of digital systems.

References

[1] K. ARNOLD and J. GOSLING. The Java Programming Language, The Java Series, Adison-Wesley, Reading, MA, 2nd edition, 1998.

[2] B. BARRAS ET AL. The Coq Proof Assistant Reference Manual - Version V6.1, Technical Report 0203, INRIA, August 1997.

[3] L. CARDELLI. In Handbook of Computer Science and Engineering, Chapter 103, pp. 2208-2236. CRC Press, 1997. http://www.research.digital.com/SRC.

[4] A. CHURCH. A formulation of the simple theory of types, *Journal of Symbolic Logic*, Vol. 5, 1940, pp. 56-68.

[5] A. CHURCH. The Calculi of Lambda-Conversion, Princeton University Press, 1941.

[6] A.A. FRAENKEL, Y. BAR-HILL, and A. LEVY. Studies in Logic and the Foundations of Mathematics, Vol. 67, North-Holland, Amsterdam, The Netherlands, second printing, second edition, 1984.

[7] J.-Y. GIRARD. Une extension de l'interprétation de Gödel á l'élimination des coupures dans l'analyse et la théorie des types, in Proceedings of the Second Scandinavian Logic Symposium, J.E. Fenstad, ed., Oslo, Norway, 1970; Studies in Logic and the Foundations of Mathematics, Vol. 63, North-Holland, Amsterdam, pp. 63-92, 1971.

[8] J.-Y. GIRARD, P. TAYLOR, and Y. LAFONT. Proof and Types, Cambridge University Press, 1989.

[9] R. HARPER, F. HONSELL, and G. PLOTKIN. A framework for defining logics, *Journal of the Association for Computing Machinery*, Vol. 40, No. 1, 1993, pp. 143-184.

[10] K. JENSEN and N. WIRTH. The Programming Language Pascal, Springer-Verlag, 1975.

[11] B.W. KERNIGHAN and D.M. RITCHIE. The C Programming Language, Second Edition, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[12] L. LAMPORT and L.C. PAULSON. Should your specification language be typed? SRC Research Report 147, Digital Systems Research Center, Palo Alto, CA, 1997. http://www.research.digital.com/SRC.

[13] M.J.C. GORDON and T.F. MELHAM. Introduction to HOL: A Theorem Proving Environment for Higher Order Logic, Cambridge University Press, 1993.

[14] B. MEYER. Eiffel: The Language. Object-oriented Series. Prentice Hall, New York, NY, 1992.

[15] R. MILNER. A theory of type polymorphism in programming, *J. Comp. Syst. Scs.*, Vol. 17, 1977, pp. 348-375.

[16] R. MILNER, M. TOFTE, and R. HARPER. The Definition of Standard ML, MIT Press, Cambridge, MA, 1991.

[17] S. OWRE, J.M. RUSHBY, and N. SHANKAR. PVS: A prototype verification system, in the Eleventh International Conference on Automated Deduction (CADE), D. Kapur, ed., Vol. 607 of Lecture Notes in Artificial Intelligence, Saratoga, NY, Springer-Verlag, pp. 748-752, 1992.

[18] L.C. PAULSON. Isabelle: A Generic Theorem Prover, Vol. 828 of Lecture Notes in Computer Science, Springer-Verlag, 1994.

[19] F.P. RAMSEY. The foundations of mathematics, in Philosophical Papers of F.P. Ramsey, D.H. Mellor, ed., Chapter 8, Cambridge University Press, Cambridge, UK, pp. 164-224, 1990. Originally published in Proceedings of the London

Mathematical Society, Vol. 25, pp. 338-384, 1925.

[20] K.H. ROSE. Explicit cycle substitutions, in Proceedings CTRS ’92 - Third International Workshop on Conditional Term Rewriting Systems, M. Rusinowitch and J.-L. Rémy, eds., No. 656 in Lecture Notes in Computer Science, Pont-a-Mousson, France, Springer-Verlag, pp. 36-50, 1992.

[21] J. RUSHBY, S. OWRE, and N. SHANKAR. Subtypes for specifications: Predicate subtyping in PVS, *IEEE Transactions on Software Engineering*, Vol. 24, No. 9, 1998, pp. 709-720.

[22] B. STROUSTRUP. The C++ Programming Language: Third Edition, Addison-Wesley Publishing Co., Reading, MA, 1997.

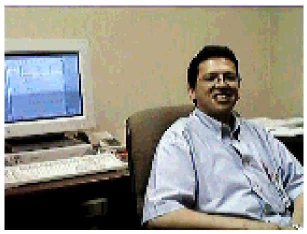
[23] A.N. WHITEHEAD and B. RUSSELL. Principia Mathematica, Cambridge University Press, Cambridge, UK, revised edition, 1925-1927. Three volumes. The first edition was published 1910-1913.

ICASE Research Quarterly
Quarterly Newsletter of the Institute for
Computer Applications in Science and Engineering

Profile

**Vol. 8, No. 4
December
1999**

César Muñoz



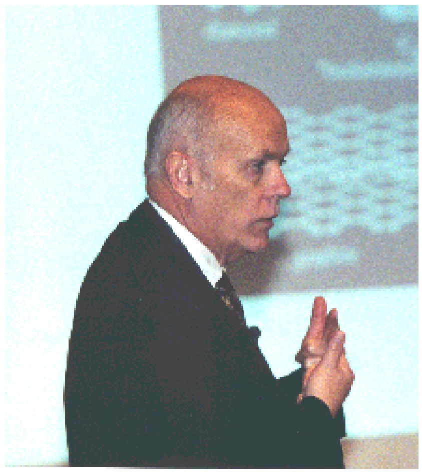
César Muñoz received his Ph.D. in Computer Science from the University of Paris 7 in November 1997. During his graduate studies he worked as a Research Assistant in the Coq Project at INRIA Rocquencourt. After completing his Ph.D., he spent one and a half years as an International Fellow in the Computer Science Laboratory at SRI International in Menlo Park. He joined ICASE as a Staff Scientist in May 1999. His research work focuses on the application of type theory and higher order logic to formal specification, verification, and proof checking.

Inside this Issue

Nobel Laureate Smalley Lectures at ICASE November 19th

**Vol. 8, No. 4
December
1999**

Inside this Issue



Professor Smalley presents seminar on "Buckytubes."

On November 19, 1999, Professor Richard E. Smalley from the Center for Nanoscale Science and Technology, Rice University, presented an ICASE Colloquium on "Buckytubes - New Materials and New Devices from Carbon." Professor Smalley is the Gene and Normal Hackerman Professor of Chemistry and Physics at Rice University. He is also the recipient of the 1996 Nobel Prize in Chemistry for his discovery of new forms of the element carbon - called fullerenes. More information on Professor Smalley and his current research can be found at http://cnst.rice.edu/reshome.html.

ICASE Colloquia
October 1, 1999 - December 31, 1999

"Launch Vehicle Design Process: Characterization, Technical Integration, and Lessons Learned"
Luke Schutzenhofer, University of Alabama, Huntsville, October 8, 1999

"Biomimetics Seminar: Swimming by Design: Fish, Whales, and Submarines"
Steven Wainwright, Duke University, November 2, 1999

"Mona Tutorial: Automaton-based Symbolic Computation"
Nils Klarlund, AT&T Labs Research, November 4, 1999

"Smart Objects and Dumb Archives: Using Buckets in Digital Libraries"
Michael L. Nelson, NASA Langley Research Center, November 9, 1999

"Buckytubes - New Materials and New Devices from Carbon"
Richard E. Smalley, Rice University, November 19, 1999

"Reduced Order Models in Unsteady Aerodynamics"
Earl Dowell, Duke University, December 7, 1999

"Fully Nonlinear Models for Internal Wave Propagation in Two-fluid Systems"
Roberto Camassa, University of North Carolina at Chapel Hill, December 10, 1999

**Vol. 8, No. 4
December
1999**

Inside this Issue

Vol. 8, No. 4
December
1999

Inside this Issue

ICASE Reports

October 1, 1999 - December 31, 1999

99-39 Rubinstein, Robert, and Ye Zhou: **Characterization of sound radiation by unresolved scales of motion in computational aeroacoustics.** ICASE Report No. 99-39, (NASA/CR-1999-209688), October 13, 1999, 13 pages.

99-40 Povitsky, Alex: **Wavefront cache-friendly algorithm for compact numerical schemes.** ICASE Report No. 99-40, (NASA/CR-1999-209708), October 15, 1999, 12 pages

99-41 Ma, Kwan-Liu, and Thomas W. Crockett: **Parallel visualization of large-scale aerodynamics calculations: A case study on the Cray T3E.** ICASE Report No. 99-41, (NASA/CR-1999-209709), October 18, 1999, 17 pages.

99-42 Lüttgen, Gerald, Michael von der Beeck, and Rance Cleaveland: **Statecharts via process algebra.** ICASE Report No. 99-42, (NASA/CR-1999-209713), October 26, 1999, 23 pages.

99-43 Muñoz, César: **Dependent types and explicit substitutions.** ICASE Report No. 99-43, (NASA/CR-1999-209722), November 5, 1999, 31 pages.

99-44 Mavriplis, Dimitri J.: **Large-scale parallel viscous flow computations using an unstructured multigrid algorithm.** ICASE Report No. 99-44, (NASA/CR-1999-209724), November 10, 1999, 19 pages.

99-45 Woodruff, S.L., J.V. Shebalin, and M.Y. Hussaini: **Direct-numerical and**

large-eddy simulations of a non-equilibrium turbulent Kolmogorov flow. ICASE Report No. 99-45, (NASA/CR-1999-209727), December 30, 1999, 21 pages.

99-46 Yamaleev, Nail K.: **Minimization of the truncation error by grid adaptation.** ICASE Report No. 99-46, (NASA/CR-1999-209729), December 2, 1999, 40 pages.

99-47 Muñoz, César: **Proof-term synthesis on dependent-type systems via explicit substitutions.** ICASE Report No. 99-47, (NASA/CR-1999-209730), November 30, 1999, 33 pages.

99-48 Povitsky, Alex: **On aeroacoustics of a stagnation flow near a rigid wall.** ICASE Report No. 99-48, (NASA/CR-1999-209825), December 6, 1999, 23 pages.

99-49 Alexandrov, Natalia M., Robert Michael Lewis, Clyde R. Gumbert, Larry L. Green, and Perry A. Newman: **Optimization with variable-fidelity models applied to wing design.** ICASE Report No. 99-49, (NASA/CR-1999-209826), December 6, 1999, 23 pages.

99-50 Ciardo, Gianfranco, Gerald Lüttgen, and Radu Siminiceanu: **Efficient symbolic state-space construction for asynchronous systems.** ICASE Report No. 99-50, (NASA/CR-1999-209827), December 13, 1999, 43 pages.

99-51 Thomas, James L., Boris Diskin, and Achi Brandt: **Textbook multigrid efficiency for the incompressible Navier-Stokes equations: High Reynolds number wakes and boundary layers.** ICASE Report No. 99-51, (NASA/CR-1999-209831), December 22, 1999, 27 pages.

99-52 Smith, Ralph C., and Zoubeida Ounaies: **A domain wall model for hysteresis in piezoelectric materials.**

ICASE Report No. 99-52, (NASA/CR-1999-209832), December 22, 1999, 21 pages.

99-53 Park, C., Z. Ounaies, J. Su, J.G. Smith, Jr., and J.S. Harrison: **Polarization stability of amorphous piezoelectric polyimides.** ICASE Report No. 99-53, (NASA/CR-1999-209833), December 22, 1999, 11 pages.

99-54 Nordström, Jan, and Mark H. Carpenter: **High order finite difference methods, multidimensional linear problems and curvilinear coordinates.** ICASE Report No. 99-54, (NASA/CR-1999-209834), December 30, 1999, 35 pages.

Employment Opportunities at ICASE

**Vol. 8, No. 4
December
1999**

Staff Scientist Position in Applied & Numerical Math

Staff Scientist Positions in Biomimetics

Staff Scientist Positions in Computer Science

Staff Scientist Position in Fluid Mechanics

Inside this Issue

**Vol. 8, No. 4
December
1999**

Inside this Issue

Publication Information

The **ICASE Research Quarterly** is published by the Institute for Computer Applications in Science and Engineering. The Institute is an affirmative action/equal opportunity employer operated by the Universities Space Research Association and is located at NASA Langley Research Center.

Editor
Shannon K. Verstynen

Editorial Assistant
Emily N. Todd

Subscription Information

You can receive e-mail notification of future editions of the **ICASE Research Quarterly** by going to the ICASE Mailing Lists Web Page and selecting the entry for the Quarterly.

Further Information

Further information about ICASE or the contents of the Research Quarterly can be obtained by contacting:

Emily Todd
ICASE
Mail Stop 132C
NASA Langley Research Center
Hampton, VA 23681-2199
Fax: (757) 864-6134
info@icase.edu

Navigation Help

Vol. 8, No. 4
December 1999

Inside this Issue

The ICASE logo, which appears on the upper left corner of the page, is a link to ICASE's Home Page.

The left and right arrows are links to the previous and next page, respectively. They are displayed on the right, upper and lower, corners of the page.

The up arrow is a quick way of returning to the top of the page. It is displayed at the bottom, left corner, of the page.

This is an image map for downloading different text formats. It appears in the Research Quarterly Reports page. The left-upper portion of the icon is a link to postscript format; the right-upper portion is a link to Adobe PDF format; the bottom-half of the icon is a link to the NCSTRL report depository. It provides an abstract of the paper and the ability to perform a field search.